Perfectly Parallel Fairness Certification of Neural Networks

CATERINA URBAN, INRIA and DIENS, École Normale Supérieure, CNRS, PSL University, France
MARIA CHRISTAKIS, MPI-SWS, Germany
VALENTIN WÜSTHOLZ, ConsenSys, Germany
FUYUAN ZHANG, MPI-SWS, Germany

Recently, there is growing concern that machine-learned software, which currently assists or even automates decision making, reproduces, and in the worst case reinforces, bias present in the training data. The development of tools and techniques for certifying fairness of this software or describing its biases is, therefore, critical. In this paper, we propose a perfectly parallel static analysis for certifying fairness of feed-forward neural networks used for classification of tabular data. When certification succeeds, our approach provides definite guarantees, otherwise, it describes and quantifies the biased input space regions. We design the analysis to be sound, in practice also exact, and configurable in terms of scalability and precision, thereby enabling pay-as-you-go certification. We implement our approach in an open-source tool called libra and demonstrate its effectiveness on neural networks trained on popular datasets.

CCS Concepts: • Software and its engineering → Formal software verification; • Theory of computation → Program analysis; Abstraction; • Computing methodologies → Neural networks; • Social and professional topics → Computing / technology policy.

Additional Key Words and Phrases: Fairness, Neural Networks, Abstract Interpretation, Static Analysis

ACM Reference Format:

1 INTRODUCTION

Due to the tremendous advances in machine learning and the vast amounts of available data, machine-learned software has an ever-increasing important role in assisting or even autonomously making decisions that impact our lives. We are already witnessing the wide adoption and societal impact of such software in criminal justice, health care, and social welfare, to name a few examples. It is not far-fetched to imagine a future where most of the decision-making is automated.

However, several studies have recently raised concerns about the fairness of such systems. For instance, consider a commercial recidivism-risk assessment algorithm that was found racially biased [Larson et al. 2016]. Similarly, a commercial algorithm that is widely used in the U.S. health care system falsely determined that Black patients were healthier than other equally sick patients by using health costs to represent health needs [Obermeyer et al. 2019]. There is also empirical evidence of gender bias in image searches, for instance, there are fewer results depicting women when searching for certain occupations, such as CEO [Kay et al. 2015]. Commercial facial recognition

Authors’ addresses: Caterina Urban, INRIA and DIENS, École Normale Supérieure, CNRS, PSL University, Paris, France, caterina.urban@inria.fr; Maria Christakis, MPI-SWS, Germany, maria@mpi-sws.org; Valentin Wüstholz, ConsenSys, Germany, valentin.wustholz@consensys.net; Fuyuan Zhang, MPI-SWS, Germany, fuyuan@mpi-sws.org.

This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2020 Copyright held by the owner/author(s).
2475-1421/2020/11-ART185
https://doi.org/10.1145/3428253

algorithms, which are increasingly used in law enforcement, are less effective for women and
darker skin types [Buolamwini and Gebru 2018].

In other words, machine-learned software may reproduce, or even reinforce, bias that is directly
or indirectly present in the training data. This awareness will certainly lead to regulations and
strict audits in the future. It is, therefore, critical to develop tools and techniques for certifying
fairness of machine-learned software or understanding the circumstances of its biased behavior.

**Feed-Forward Neural Networks.** We make a step forward in meeting these needs by designing
a novel static analysis framework for certifying fairness of feed-forward neural networks used for
classification of tabular data (e.g., stored in Excel files or relational databases).

While, in general, machine learning approaches such as random forests or gradient boosting
are preferred over neural networks for analyzing tabular data, for some datasets, feed-forward
neural networks provide better accuracy and flexibility at the cost of putting slightly more effort
into hyperparameter optimizations. This is the case, for instance, for very large datasets (i.e., with
billions of rows) or data that comes in batches over time. As these kinds of datasets are becoming
more and more common, so are neural network-based decision-making software systems.

The other challenge is that neural networks are harder to interpret than such machine learning
approaches and, thus, it becomes harder to ensure that neural network-based decision-making
software is doing what intended. This is precisely what our work aims to address.

**Dependency Fairness.** Our static analysis framework is designed for certifying dependency
Specifically, given a choice (e.g., driven by a causal model [Pearl 2009] or a correlation analysis)
of input features that are considered (directly or indirectly) sensitive to bias, a neural network is
fair if the output classification is not affected by different values of the chosen features. We chose this
definition over other fairness notions because it does not require an oracle and it is specifically
designed to be testable and, thus, also amenable to static analysis. It is also a stronger notion than
group-based fairness notions (see Section 12 for a more in-depth comparison).

Of course, an obvious way to avoid dependencies between the classification and the sensitive
features is to entirely remove the sensitive features from the training data [Grgić-Hlača et al. 2016].
However, this in general does not work for three main reasons. First, neural networks learn from
latent variables (e.g., [Lum and Isaac 2016; Udeshi et al. 2018]). For instance, a credit-screening
algorithm that does not take race (or gender) as an input might still be biased with respect to it,
say, by using the ZIP code of applicants as proxy for race (or their first name as proxy for gender).
Therefore, simply removing a sensitive feature does not necessarily free the training data or the
corresponding trained neural network from bias. Second, the training data is only a relatively small
sample of the entire input space, on portions of which the neural network might end up being
biased even if trained with fair data. For example, if women are under-represented in the training
data, this leaves freedom to the training process and thus the resulting classifier might end up being
biased against them. Third, the information provided by a sensitive feature might be necessary for
business necessity [Barocas and Selbst 2016], for instance, to introduce intended bias in a certain
input region. Assume a credit-screening model that should discriminate with respect to age only
above a particular threshold. Above this age threshold, the higher the requested credit amount, the
lower the chances of receiving it. In such cases, removing the sensitive feature is not even possible.
**Our Approach.** Static analysis of neural networks is still in its infancy. Most of the work in the area focuses on certifying local robustness of neural networks against adversarial examples [Gehr et al. 2018; Huang et al. 2017; Singh et al. 2019, etc.] or proving functional properties of neural networks [Katz et al. 2017; Wang et al. 2018, etc.]. On the other hand, dependency fairness is a global property, relative to the entire input space, instead of only those inputs within a particular distance metric or that satisfy a given functional specification. It is thus much harder to verify. The approach that we propose in this paper brings us closer to the aspiration for a practical general verification approach of global neural network properties.

Our approach certifies dependency fairness of feed-forward neural networks by employing a combination of a forward and a backward static analysis. On a high level, the forward pass aims to reduce the overall analysis effort. At its core, it divides the input space of the network into independent partitions. The backward analysis then attempts to certify fairness of the classification within each partition (in a perfectly parallel fashion) with respect to a chosen (set of) feature(s), which may be directly or indirectly sensitive, for instance, race or ZIP code. In the end, our approach reports for which regions of the input space the neural network is certified to be fair and for which there is bias. Note that we do not necessarily need to analyze the entire input space; our technique is also able to answer specific bias queries about a fraction of the input space, e.g., are Hispanics over 45 years old discriminated against with respect to gender?

The scalability-vs-precision tradeoff of our approach is configurable. Partitions that do not satisfy the given configuration are excluded from the analysis and may be resumed later, with a more flexible configuration. This enables usage scenarios in which our approach adapts to the available resources, e.g., time or CPUs, and is run incrementally. In other words, we designed a pay-as-you-go certification approach that the more resources it is given, the larger the region of the input space it is able to analyze.

**Related Work.** In the literature, related work on determining fairness of machine-learning models has focused on providing probabilistic guarantees [Bastani et al. 2019]. In contrast, our approach gives definite guarantees for those input partitions that satisfy the analysis configuration. Similarly to our approach, there is work that also aims to provide definite guarantees [Albarghouthi et al. 2017b] (although for different fairness criteria). However, it has been shown to scale only up to neural networks with two hidden neurons. We demonstrate that our approach can effectively analyze networks with hundreds (or, in some cases, even thousands) of hidden neurons. Ours is also the first approach in this area that is configurable in terms of scalability and precision.

**Contributions.** We make the following contributions:

1. We propose a novel perfectly parallel static analysis approach for certifying dependency fairness of feed-forward neural networks used for classification of tabular data. If certification fails, our approach can describe and quantify the biased input space region(s).
2. We rigorously formalize our approach and show that it is sound and, in practice, exact for the analyzed regions of the input space.
3. We discuss the configurable scalability-vs-precision tradeoff of our approach that enables pay-as-you-go certification.
4. We implement our approach in an open-source tool called LIBRA and extensively evaluate it on neural networks trained with popular datasets. We show the effectiveness of our approach in detecting injected bias and answering bias queries. We also experiment with the precision and scalability of the analysis and discuss the tradeoffs.

2 OVERVIEW

In this section, we give an overview of our approach using a small constructed example, which is shown in Figure 1.

**Example.** The figure depicts a feed-forward neural network for credit approval. There are two inputs $x_{0,1}$ and $x_{0,2}$ (shown in purple). Input $x_{0,1}$ denotes the requested credit amount and $x_{0,2}$ denotes age. Both inputs have continuous values in the range $[0, 1]$. Output $x_{3,2}$ (shown in green) denotes that the credit request is approved, whereas $x_{3,1}$ (in red) denotes that it is denied. The neural network also consists of two hidden layers with two nodes each (in gray).

Now, let us assume that this neural network is trained to deny requests for large credit amounts from older people. Otherwise, the network should not discriminate with respect to age for small credit amounts. There should also not be discrimination for younger people with respect to the requested credit amount. When choosing age as the sensitive input, our approach can certify fairness with respect to different age groups for small credit amounts. Our approach is also able to find (as well as quantify) bias with respect to age for large credit amounts. Note that, in general, bias found by the analysis may be intended or accidental — our analysis does not aim to address this question. It is up to the user to decide whether the result of the analysis is expected or not.

Our approach does not require age to be an input of the neural network. For example, $x_{0,2}$ could denote the ZIP code of credit applicants, and the network could still use it as proxy for age. That is, requests for large credit amounts are denied for a certain range of ZIP codes (where older people tend to live), yet there is no discrimination between ZIP codes for small credit amounts. When choosing the ZIP code as the sensitive input, our approach would similarly be able to detect bias with respect to it for large credit amounts. In general, the choice of the sensitive features is up to the user, e.g., it can be driven by knowledge of proxies coming from a separate correlation analysis.

Below, we present on a high level how our approach achieves these results.

**Naïve Approach.** In theory, the simplest way to certify dependency fairness is to first analyze the neural network backwards starting from each output node, in our case $x_{3,1}$ and $x_{3,2}$. This allows us to determine the regions of the input space (i.e., age and requested credit amount) for which credit is approved and denied. For example, assume that we find that requests are denied for credit amounts larger than 10 000 (i.e., $10 000 < x_{0,1}$) and age greater than 60 (i.e., $60 < x_{0,2}$), while they are approved for $x_{0,1} \leq 10 000$ and $60 < x_{0,2}$ or for $x_{0,2} \leq 60$.

The second step is to forget the value of the sensitive input (i.e., age) or, in other words, to project these regions over the credit amount. In our example, after projection we have that credit requests are denied for $10 000 < x_{0,1}$ and approved for any value of $x_{0,1}$. A non-empty intersection between the projected input regions indicates bias with respect to the sensitive input. In our example, the intersection is non-empty for $10 000 < x_{0,1}$: there exist people that differ in age but request the same credit amount (greater than 10 000), some of whom receive the credit while others do not.
This approach, however, is not practical. Specifically, in neural networks with ReLU activation functions (see Section 3 for more details, other activation functions are discussed in Section 9), each hidden node effectively represents a disjunction between two activation statuses (active and inactive). In our example, there are $2^{4}$ possible activation patterns for the 4 hidden nodes. To retain maximum precision, the analysis would have to explore all of them, which does not scale in practice.

Our Approach. Our analysis is based on the observation that there might exist many activation patterns that do not correspond to a region of the input space [Hanin and Rolnick 2019]. Such patterns can, therefore, be ignored during the analysis. We push this idea further by defining abstract activation patterns, which fix the activation status of only certain nodes and thus represent sets of (concrete) activation patterns. Typically, a relatively small number of abstract activation patterns is sufficient for covering the entire input space, without necessarily representing and exploring all possible concrete patterns.

Identifying those patterns that definitely correspond to a region of the input space is only possible with a forward analysis. Hence, we combine a forward pre-analysis with a backward analysis. The pre-analysis partitions the input space into independent partitions corresponding to abstract activation patterns. Then, the backward analysis tries to prove fairness of the neural network for each such partition.

More specifically, we set an upper bound $U$ on the number of tolerated disjunctions (i.e., on the number of nodes with an unknown activation status) per abstract activation pattern. Our forward pre-analysis uses a cheap abstract domain (e.g., the boxes domain [Cousot and Cousot 1976]) to iteratively partition the input space along the non-sensitive input dimensions to obtain fair input partitions (i.e., boxes). Each partition satisfies one of the following conditions: (a) its classification is already fair because only one network output is reachable for all inputs in the region, (b) it has an abstract activation pattern with at most $U$ unknown nodes, or (c) it needs to be partitioned further. We call partitions that satisfy condition (b) feasible.

In our example, let $U = 2$. At first, the analysis considers the entire input space, that is, $x_{0,1} : [0, 1]$ (credit amount) and $x_{0,2} : [0, 1]$ (age). (Note that we could also specify a subset of the input space for analysis.) The abstract activation pattern corresponding to this initial partition $I$ is $\epsilon$ (i.e., no hidden nodes have fixed activation status) and, thus, the number of disjunctions would be 4, which is greater than $U$. Therefore, I needs to be divided into $I_{1}$ ($x_{0,1} : [0, 0.5], x_{0,2} : [0, 1]$) and $I_{2}$ ($x_{0,1} : [0.5, 1], x_{0,2} : [0, 1]$). Observe that the input space is not split with respect to $x_{0,2}$, which is the sensitive input. Now, $I_{1}$ is feasible since its abstract activation pattern is $x_{1,2}x_{2,1}$ (i.e., 3 nodes are always active), while $I_{2}$ must be divided further since its abstract activation pattern is $\epsilon$.

To control the number of partitions, we impose a lower bound $L$ on the size of each of their dimensions. Partitions that require a dimension of a smaller size are excluded. In other words, they are not considered until more analysis budget becomes available, that is, a larger $U$ or a smaller $L$.

In our example, let $L = 0.25$. The forward pre-analysis further divides $I_{2}$ into $I_{2,1}$ ($x_{0,1} : [0.5, 0.75], x_{0,2} : [0, 1]$) and $I_{2,2}$ ($x_{0,1} : [0.75, 1], x_{0,2} : [0, 1]$). Now, $I_{2,1}$ is feasible, with abstract pattern $x_{1,2}x_{2,1}$, while $I_{2,2}$ is not. However, $I_{2,2}$ may not be split further because the size of the only non-sensitive dimension $x_{0,1}$ has already reached the lower bound $L$. As a result, $I_{2,2}$ is excluded, and only the remaining 75% of the input space is considered for analysis.

Next, feasible input partitions (within bounds $L$ and $U$) are grouped by abstract activation patterns. In our example, the pattern corresponding to $I_{1}$, namely $x_{1,2}x_{2,1}$, is subsumed by the (more abstract) pattern of $I_{2,1}$, namely $x_{1,2}x_{2,1}$. Consequently, we group $I_{1}$ and $I_{2,1}$ under pattern $x_{1,2}x_{2,1}$.

The backward analysis is then run in parallel for each representative abstract activation pattern, in our example $x_{1,2}x_{2,1}$. This analysis determines the region of the input space (within a given
partition group) for which each output of the neural network is returned, e.g., credit is approved for \( c_1 \leq x_{0,1} \leq c_2 \) and \( a_1 \leq x_{0,2} \leq a_2 \). To achieve this, the analysis uses an expensive abstract domain, for instance, disjunctive or powerset polyhedra [Cousot and Cousot 1979; Cousot and Halbwachs 1978], and leverages abstract activation patterns to avoid disjunctions. For instance, pattern \( x_{1,2}x_{2,1} \) only requires reasoning about two disjunctions from the remaining hidden nodes \( x_{1,1} \) and \( x_{2,2} \).

Finally, fairness is checked for each partition in the same way that it is done by the naive approach for the entire input space. In our example, we prove that the classification within \( I_1 \) is fair and determine that within \( I_{2,1} \) the classification is biased. Concretely, our approach determines that bias occurs for \( 0.54 \leq x_{0,1} \leq 0.75 \), which corresponds to \( 21\% \) of the entire input space (assuming a uniform probability distribution). In other words, the network returns different outputs for people that request the same credit in the above range but differ in age. Recall that partition \( I_{2,2} \), where \( 0.75 \leq x_{0,1} \leq 1 \), was excluded from analysis, and therefore, we cannot draw any conclusions about whether there is any bias for people requesting credit in this range.

Note that bias may also be quantified according to a probability distribution of the input space. In particular, it might be that credit requests in the range \( 0.54 \leq x_{0,1} \leq 0.75 \) are more (resp. less) common in practice. Given their probability distribution, our analysis computes a tailored percentage of bias, which in this case would be greater (resp. less) than \( 21\% \).

3 FEED-FORWARD DEEP NEURAL NETWORKS

Formally, a feed-forward deep neural network consists of an input layer \((l_0)\), an output layer \((l_N)\), and a number of hidden layers \((l_1, \ldots, l_{N-1})\) in between. Each layer \( l_i \) contains \(|l_i|\) nodes and, with the exception of the input layer, is associated to a \(|l_i| \times |l_{i-1}|\)-matrix \( W_i \) of weight coefficients and a vector \( B_i \) of \(|l_i|\) bias coefficients. In the following, we use \( X \) to denote the set of all nodes, \( X_i \) to denote the set of nodes of the \( i \)th layer, and \( x_{i,j} \) to denote the \( j \)th node of the \( i \)th layer of a neural network. We focus here on neural networks used for classification tasks. Thus, \(|l_N|\) is the number of target classes (e.g., 2 classes in Figure 1).

The value of the input nodes is given by the input data: continuous data is represented by one input node (e.g., \( x_{0,1} \) or \( x_{0,2} \) in Figure 1), while categorical data is represented by multiple input nodes via one-hot encoding. In the following, we use \( K \) to denote the subset of input nodes considered (directly or indirectly) sensitive to bias (e.g., \( x_{0,2} \) in Figure 1) and \( \overline{K} \) is the number of target classes minus the set of input nodes not deemed sensitive to bias.

The value of the hidden and output node \( x_{i,j} \) is computed by an activation function applied to a linear combination of the values of all nodes in the preceding layer [Goodfellow et al. 2016], i.e., \( x_{i,j} = f(\sum_{k=1}^{l_i-1} w_{j,k}^i \cdot x_{i-1,k} + b_{i,j}) \), where \( w_{j,k}^i \) and \( b_{i,j} \) are weight and bias coefficients in \( W_i \) and \( B_i \), respectively. In a fully-connected neural network, all \( w_{j,k}^i \) are non-zero. Weights and biases are adjusted during the training phase of the neural network. In what follows, we focus on already trained neural networks, which we call neural-network models.

Nowadays, the most commonly used activation for hidden nodes is the Rectified Linear Unit (ReLU) [Nair and Hinton 2010]: \( \text{ReLU}(x) = \max(x, 0) \). In this case, the activation used for output nodes is the identity function. The output values are then normalized into a probability distribution on the target classes [Goodfellow et al. 2016]. We discuss other activation functions in Section 9.

4 TRACE SEMANTICS

In the following, we represent neural-network models as sequential programs. These programs consist of assignments for computing the activation value of each node (e.g., \( x_{1,1} = -0.31 \cdot x_{0,1} + 0.99 \cdot x_{0,2} - 0.63 \) in Figure 1) and implementations of activation functions (e.g., if-statements for
RELU). As is standard practice in static program analysis, we define a semantics for these programs that is tailored to our property of interest, dependency fairness, and use it to prove soundness of our approach. In particular, this allows us to formally prove soundness of the parallelization of our analysis as well (cf. Section 9), unlike for most of the existing approaches in which the parallelization is only introduced as part of the implementation and never proven sound.

The semantics of a neural-network model is a mathematical characterization of its behavior when executed for all possible input data. We model the operational semantics of a feed-forward neural-network model $M$ as a transition system $\langle \Sigma, \tau \rangle$, where $\Sigma$ is a (finite but exceedingly large) set of states and the acyclic transition relation $\tau \subseteq \Sigma \times \Sigma$ describes the possible transitions between states [Cousot 2002; Cousot and Cousot 1977].

More specifically, a state $s \in \Sigma$ maps neural-network nodes to their values. Here, for simplicity, we assume that nodes have real values, i.e., $s : X \rightarrow \mathbb{R}$. (We discuss floating-point values in Section 9.) In the following, we often only care about the values of a subset of the neural-network nodes in certain states. Thus, let $\Sigma|_Y \defeq \{ s_Y \mid s \in \Sigma \}$ be the restriction of $\Sigma$ to a domain of interest $Y$. Sets $\Sigma|_{\mathbb{X}_0}$ and $\Sigma|_{\mathbb{X}_n}$ denote restrictions of $\Sigma$ to the network nodes in the input and output layer, respectively. With a slight abuse of notation, let $X_{i,j}$ denote $\Sigma|\{(x_{i,j})\}$, i.e., the restriction of $\Sigma$ to the singleton set containing $x_{i,j}$. Transitions happen between states with different values for consecutive nodes in the same layer, i.e., $\tau \subseteq X_{i,j} \times X_{i,j+1}$, or between states with different values for the last and first node of consecutive layers of the network, i.e., $\tau \subseteq X_{i,i+1} \times X_{i+1,0}$. The set $\Omega \defeq \{ s \in \Sigma \mid \forall s' \in \Sigma : \langle s, s' \rangle \not\in \tau \}$ is the set of final states of the neural network. These are partitioned in a set of outcomes $\bigcup_{n \in \mathbb{N}} S^n \defeq \{ s \in \Sigma \mid \max X_{n,i} = x_{n,i} \mid 0 \leq i \leq |\Omega_n| \}$, depending on the output node with the highest value (i.e., the target class with highest probability).

Let $\Sigma^n \defeq \{ s_0 \cdots s_{n-1} \mid \forall i < n : s_i \in \Sigma \}$ be the set of all sequences of exactly $n$ states in $\Sigma$. Let $\Sigma^+ \defeq \bigcup_{n \in \mathbb{N}} S^n$ be the set of all non-empty finite sequences of states. A trace is a sequence of states that respects the transition relation $\tau$, i.e., $\langle s, s' \rangle \in \tau$ for each pair of consecutive states $s, s'$ in the sequence. We write $\Sigma^n$ for the set of all traces of $n$ states: $\Sigma^n \defeq \{ s_0 \cdots s_{n-1} \in \Sigma^n \mid \forall i < n-1 : \langle s_i, s_{i+1} \rangle \in \tau \}$. The trace semantics $\Upsilon \in \mathcal{P}(\Sigma^+)$ generated by a transition system $\langle \Sigma, \tau \rangle$ is the set of all non-empty traces terminating in $\Omega$ [Cousot 2002]:

$$\Upsilon \defeq \bigcup_{n \in \mathbb{N}^+} \{ s_0 \ldots s_{n-1} \in \Sigma^n \mid s_{n-1} \in \Omega \}$$

In the rest of the paper, we write $[M]$ to denote the trace semantics of a neural-network model $M$.

The trace semantics fully describes the behavior of $M$. However, reasoning about a certain property of $M$ does not need all this information and, in fact, is facilitated by the design of a semantics that abstracts away from irrelevant details. In the following sections, we formally define our property of interest, dependency fairness, and systematically derive, using abstract interpretation [Cousot and Cousot 1977], a semantics tailored to reasoning about this property.

### 5 Dependency Fairness

A property is specified by its extension, that is, by the set of elements having such a property [Cousot and Cousot 1977, 1979]. Properties of neural-network models are properties of their semantics. Thus, properties of network models with trace semantics in $\mathcal{P}(\Sigma^+)$ are sets of sets of traces in $\mathcal{P}(\Sigma^+)$. In particular, the set of neural-network properties forms a complete boolean lattice $\langle \mathcal{P}(\Sigma^+) \rangle$, $\subseteq, \bigcup, \cap, \emptyset, \mathcal{P}(\Sigma^+) \rangle$ for subset inclusion, that is, logical implication. The strongest property is the standard collecting semantics $\Lambda \in \mathcal{P}(\Sigma^+)$:

$$\Lambda \defeq \{ \Upsilon \}$$
Let \( \langle M \rangle \) denote the collecting semantics of a particular neural-network model \( M \). Then, model \( M \) satisfies a given property \( \mathcal{H} \) if and only if its collecting semantics is a subset of \( \mathcal{H} \):

\[
M \models \mathcal{H} \iff \langle M \rangle \subseteq \mathcal{H}
\]  

(3)

Here, we consider the property of dependency fairness, which expresses that the classification determined by a network model does not depend on sensitive input data. In particular, the property might interest the classification of all or just a fraction of the input space.

More formally, let \( \mathcal{V} \) be the set of all possible value choices for all sensitive input nodes in \( K \), e.g., for \( K = \{ x_{0,1}, x_{0,2} \} \) one-hot encoding, say, gender information, \( \mathcal{V} = \{ \{1, 0\}, \{0, 1\} \} \); for \( K = \{ x_{0,k} \} \) encoding continuous data, say, in the range \( [0, 1] \), we use binning so a possibility is, e.g., \( \mathcal{V} = \{ [0, 0.25], [0.25, 0.75], [0.75, 1] \} \). In the following, given a trace \( \sigma \in \mathcal{P}(\Sigma^+) \), we write \( \sigma_0 \) and \( \sigma_\omega \) to denote its initial and final state, respectively. We also write \( \sigma_0 =_K \sigma'_0 \) to indicate that the states \( \sigma_0 \) and \( \sigma'_0 \) agree on all values of all non-sensitive input nodes, and \( \sigma_\omega =_K \sigma'_\omega \) to indicate that \( \sigma_0 \) and \( \sigma'_0 \) have the same outcome \( O \in \mathcal{O} \). We can now formally define when the sensitive input nodes in \( K \) are unused with respect to a set of traces \( T \in \mathcal{P}(\Sigma^+) \) [Urban and Müller 2018]. For one-hot encoded sensitive inputs\(^2\), we have

\[
\text{unused}_K(T) \overset{\text{def}}{=} \forall \sigma \in T, \forall \mathcal{V} \in \mathcal{V} : \sigma_0(K) \neq \mathcal{V} \Rightarrow \exists \sigma' \in T : \sigma_0 =_K \sigma'_0 \wedge \sigma'_0(K) = \mathcal{V} \wedge \sigma_\omega =_K \sigma'_\omega,
\]

(4)

where \( \sigma_0(K) =_K \{ \sigma_0(x) \mid x \in K \} \) is the image of \( K \) under \( \sigma_0 \). Intuitively, the sensitive input nodes in \( K \) are unused if any possible outcome in \( T \) (i.e., any outcome \( \sigma_\omega \) of any trace \( \sigma \) in \( T \)) is possible from all possible value choices for \( K \) (i.e., there exists a trace \( \sigma' \) in \( T \) for each value choice for \( K \) with the same outcome as \( \sigma \)). That is, each outcome is independent of the value choice for \( K \).

**Example 5.1.** Let us consider again our example in Figure 1. We write \( (c, a) \rightsquigarrow o \) for a trace starting in a state with \( x_{0,1} = c \) and \( x_{0,2} = a \) and ending in a state where \( o \) is the node with the highest value (i.e., the output class). The sensitive input \( x_{0,2} \) (age) is unused in \( T = \{ (0.5, a) \rightsquigarrow x_{3,2} \mid 0 \leq a \leq 1 \} \). It is instead used in \( T' = \{ (0.75, a) \rightsquigarrow x_{3,2} \mid 0 \leq a < 0.51 \} \cup \{ (0.75, a) \rightsquigarrow x_{3,1} \mid 0.51 \leq a \leq 1 \} \).

The dependency-fairness property \( \mathcal{F}_K \) can now be defined as \( \mathcal{F}_K \overset{\text{def}}{=} \{ \langle M \rangle \mid \text{unused}_K(\langle M \rangle) \} \), that is, as the set of all neural-network models (or rather, their semantics) that do not use the values of the sensitive input nodes for classification. In practice, the property might interest just a fraction of the input space, i.e., we define

\[
\mathcal{F}_K[Y] \overset{\text{def}}{=} \{ \langle M \rangle^Y \mid \text{unused}_K(\langle M \rangle^Y) \},
\]

(5)

where \( Y \in \mathcal{P}(\Sigma) \) is a set of initial states of interest and the restriction \( T^Y \overset{\text{def}}{=} \{ \sigma \in T \mid \sigma_0 \in Y \} \) only contains traces of \( T \in \mathcal{P}(\Sigma^+) \) that start with a state in \( Y \). Similarly, in the rest of the paper, we write \( S^Y \overset{\text{def}}{=} \{ T^Y \mid T \in S \} \) for the set of sets of traces restricted to initial states in \( Y \). Thus, from Equation 3, we have the following:

**Theorem 5.2.** \( M \models \mathcal{F}_K[Y] \iff \langle M \rangle^Y \subseteq \mathcal{F}_K[Y] \)

Proof. The proof follows trivially from Equation 3 and the definition of \( \mathcal{F}_K[Y] \) (cf. Equation 5) and \( \langle M \rangle^Y \).

\( \square \)

### 6 DEPENDENCY SEMANTICS

We now use abstract interpretation to systematically derive, by successive abstractions of the collecting semantics \( \Lambda \), a sound and complete semantics \( \Lambda_{\infty} \), that contains only and exactly the information needed to reason about \( \mathcal{F}_K[Y] \).

\( ^2\)For continuous sensitive inputs, we can replace \( \sigma_0(K) \neq V \) (resp. \( \sigma'_0(K) = V \)) with \( \sigma_0(K) \not\subseteq V \) (resp. \( \sigma'_0(K) \subseteq V \)).
6.1 Outcome Semantics

Let \( T_Z \overset{\text{def}}{=} \{ \sigma \in T \mid \sigma_{\omega} \in Z \} \) be the set of traces of \( T \in \mathcal{P} (\Sigma^+) \) that end with a state in \( Z \in \mathcal{P} (\Sigma) \). As before, we write \( S_Z \overset{\text{def}}{=} \{ T_Z \mid T \in S \} \) for the set of sets of traces restricted to final states in \( Z \). From the definition of \( \mathcal{F}_K[Y] \) (and in particular, from the definition of \( \text{UNUSED}_K \), cf. Equation 4), we have:

**Lemma 6.1.** \( (\mathcal{M})^Y \subseteq \mathcal{F}_K[Y] \) \( \iff \forall O \in \mathcal{O} : (\mathcal{M})^Y_O \subseteq \mathcal{F}_K[Y] \)

Proof. Let \( (\mathcal{M})^Y \subseteq \mathcal{F}_K[Y] \). From the definition of \( (\mathcal{M})^Y \) (cf. Equation 2), we have that \( [[\mathcal{M}]]^Y \in \mathcal{F}_K[Y] \). Thus, from the definition of \( \mathcal{F}_K[Y] \) (cf. Equation 5), we have \( \text{UNUSED}_K([[\mathcal{M}]]^Y) \). Now, from the definition of \( \text{UNUSED}_K \) (cf. Equation 4), we equivalently have \( \forall O \in \mathcal{O} : \text{UNUSED}_K([[\mathcal{M}]]^Y) \). Thus, we can conclude that \( \forall O \in \mathcal{O} : (\mathcal{M})^Y_O \subseteq \mathcal{F}_K[Y] \). \( \square \)

In particular, this means that in order to determine whether a neural-network model \( \mathcal{M} \) satisfies dependency fairness, we can independently verify, for each of its possible target classes \( O \in \mathcal{O} \), that the values of its sensitive input nodes are unused.

We use this insight to abstract the collecting semantics \( \Lambda \) by partitioning. More specifically, let \( \bullet \overset{\text{def}}{=} \{ \Sigma^+_O \mid O \in \mathcal{O} \} \) be a trace partition with respect to outcome. We have the following Galois connection

\[
(\mathcal{P} (\mathcal{P} (\Sigma^+)), \subseteq) \xrightarrow{Y} (\mathcal{P} (\mathcal{P} (\Sigma^+)), \subseteq), \tag{6}
\]

where \( \alpha_\bullet(S) \overset{\text{def}}{=} \{ T_O \mid T \in S \land O \in \mathcal{O} \} \). The order \( \subseteq \) is the pointwise ordering between sets of traces with the same outcome, i.e., \( A \subseteq B \overset{\text{def}}{=} \bigwedge_{O \in \mathcal{O}} A_O \subseteq B_O \), where \( S_Z \) denotes the only non-empty set of traces in \( S_Z \). We can now define the outcome semantics \( \Lambda_\bullet \in \mathcal{P} (\mathcal{P} (\Sigma^+)) \) by abstraction of \( \Lambda \):

\[
\Lambda_\bullet \overset{\text{def}}{=} \alpha_\bullet(\Lambda) = \{ Y_O \mid O \in \mathcal{O} \} \tag{7}
\]

In the rest of the paper, we write \( (\mathcal{M})_\bullet \) to denote the outcome semantics of a particular neural-network model \( \mathcal{M} \).

6.2 Dependency Semantics

We observe that, to reason about dependency fairness, we do not need to consider all intermediate computations between the initial and final states of a trace. Thus, we can further abstract the outcome semantics into a set of dependencies between initial states and outcomes of traces.

To this end, we define the following Galois connection\(^3\)

\[
(\mathcal{P} (\mathcal{P} (\Sigma^+)), \subseteq) \xrightarrow{Y} (\mathcal{P} (\mathcal{P} (\Sigma \times \Sigma)), \subseteq), \tag{8}
\]

where \( \alpha_{\rightarrow}(S) \overset{\text{def}}{=} \{ \{ \sigma_0, \sigma_\omega \} \mid \sigma \in T \mid T \in S \} \) [Urban and Müller 2018] abstracts away all intermediate states of any trace. We finally derive the dependency semantics \( \Lambda_{\rightarrow} \in \mathcal{P} (\mathcal{P} (\Sigma \times \Sigma)) \):

\[
\Lambda_{\rightarrow} \overset{\text{def}}{=} \alpha_{\rightarrow}(\Lambda_\bullet) = \{ \{ \sigma_0, \sigma_\omega \} \mid \sigma \in Y_O \mid O \in \mathcal{O} \} \tag{9}
\]

In the following, let \( (\mathcal{M})_{\rightarrow} \) denote the dependency semantics of a particular network model \( \mathcal{M} \).

Let \( R^Y \overset{\text{def}}{=} \{ \{ s, _ \} \mid s \in Y \} \) restrict a set of pairs of states to pairs whose first element is in \( Y \) and, similarly, let \( S^Y \overset{\text{def}}{=} \{ R^Y \mid R \in S \} \) restrict a set of sets of pairs of states to first elements in \( Y \). The next result shows that \( \Lambda_{\rightarrow} \) is sound and complete for proving dependency fairness:

**Theorem 6.2.** \( \mathcal{M} \models \mathcal{F}_K[Y] \iff (\mathcal{M})^Y_{\rightarrow} \subseteq \alpha_{\rightarrow}(\mathcal{F}_K[Y]) \)

\(^3\)Note that here and in the following, for convenience, we abuse notation and reuse the order symbol \( \subseteq \) defined over sets of sets of traces, instead of its abstraction, defined over sets of sets of pairs of states.
Proof. Let \( M \models \mathcal{F}_K[Y] \). From Theorem 5.2, we have that \( \langle M \rangle^Y \subseteq \mathcal{F}_K[Y] \). Thus, from the Galois connections in Equation 6 and 8, we have \( \alpha_\omega(\alpha_L(\langle M \rangle^Y)) \subseteq \alpha_\omega(\mathcal{F}_K[Y]) \). From the definition of \( \langle M \rangle^Y_\omega \) (cf. Equation 9), we can then conclude that \( \langle M \rangle^Y_\omega \subseteq \alpha_\omega(\mathcal{F}_K[Y]) \). \( \square \)

**Corollary 6.3.** \( M \models \mathcal{F}_K[Y] \iff \langle M \rangle^Y_\omega \subseteq \alpha_\omega(\mathcal{F}_K[Y]) \)

**Proof.** The proof follows trivially from the definition of \( \subseteq \) (cf. Equation 6 and 8) and Lemma 6.1. \( \square \)

Furthermore, we observe that partitioning with respect to outcome induces a partition of the space of values of the input nodes used for classification. For instance, partitioning \( T \) in Example 5.1 induces a partition on the values of (the indeed used node) \( x_{0.2} \). Thus, we can equivalently verify whether \( \langle M \rangle^Y_\omega \subseteq \alpha_\omega(\mathcal{F}_K[Y]) \) by checking if the dependency semantics \( \langle M \rangle^Y_\omega \) induces a partition of \( Y_K \). Let \( R_{0} \) \( \equiv \{ s \mid \langle \_ , \_ \rangle \in R \} \) (resp. \( R_{\omega} \) \( \equiv \{ s \mid \langle \_ , \_ \rangle \in R \} \) be the selection of the first (resp. last) element from each pair in a set of pairs of states. We formalize this observation below.

**Lemma 6.4.** \( M \models \mathcal{F}_K[Y] \iff \forall A, B \in \langle M \rangle^Y_\omega : (A_{\omega} \neq B_{\omega} \Rightarrow A_{[Y]} \cap B_{[Y]} = \emptyset) \)

**Proof.** Let \( M \models \mathcal{F}_K[Y] \). From Corollary 6.3, we have that \( \langle M \rangle^Y_\omega \subseteq \alpha_\omega(\mathcal{F}_K[Y]) \). Thus, from the definition of \( \langle M \rangle^Y_\omega \) (cf. Equation 9), we have \( \forall O \in \Theta : \alpha_\omega(\langle M \rangle^Y_\Theta) \subseteq \alpha_\omega(\mathcal{F}_K[Y]) \). In particular, from the definition of \( \alpha_\omega \) and \( \mathcal{F}_K[Y] \) (cf. Equation 5), we have that \( \text{UNUSED}_K(\langle M \rangle^Y_\Theta) \) for each \( O \in \Theta \). From the definition of \( \text{UNUSED}_K \) (cf. Equation 4), for each pair of non-empty \( \langle M \rangle^Y_\Theta \) and \( \langle M \rangle^Y_\Theta \) for different \( O_1, O_2 \in \Theta \) (the case in which one or both are empty is trivial), it must necessarily be the value of the non-sensitive input nodes in \( K \) that causes the different outcome \( O_1 \) or \( O_2 \). We can thus conclude that \( \forall A, B \in \langle M \rangle^Y_\omega : (A_{\omega} \neq B_{\omega} \Rightarrow A_{[Y]} \cap B_{[Y]} = \emptyset) \). \( \square \)

**7 NAÏVE DEPENDENCY-FAIRNESS ANALYSIS**

In this section, we present a first static analysis for dependency fairness that computes a sound over-approximation \( \Lambda^b_\omega \) of the dependency semantics \( \Lambda_\omega \), i.e., \( \Lambda_\omega \subseteq \Lambda^b_\omega \). This analysis corresponds to the naïve approach we discussed in Section 2. While it is too naïve to be practical, it is still useful for building upon later in the paper.

For simplicity, we consider ReLU activation functions. (We discuss extensions to other activation functions in Section 9.) The naïve static analysis is described in Algorithm 1. It takes as input (cf. Line 14) a neural-network model \( M \), a set of sensitive input nodes \( K \) of \( M \), a (representation of a) set of initial states of interest \( Y \), and an abstract domain \( A \) to be used for the analysis. The analysis proceeds backwards for each outcome (i.e., each target class \( x_{n,j} \)) of \( M \) (cf. Line 17) in order to determine an over-approximation of the initial states that satisfy \( Y \) and lead to \( x_{n,j} \) (cf. Line 18).

More specifically, the transfer function \( \text{OUTCOME}_A[\cdot] \) (cf. Line 2) modifies a given abstract-domain element to assume the given outcome \( x \), that is, to assume that \( \max X_N = x \). The transfer functions \( \text{RELU}_A[\cdot] \) and \( \text{ASSIGN}_A[\cdot] \) (cf. Line 5) respectively consider a ReLU operation and replace \( x_{i,j} \) with the corresponding linear combination of nodes in the preceding layer (see Section 3).

Finally, the analysis checks whether the computed over-approximations satisfy dependency fairness with respect to \( K \) (cf. Line 19). In particular, it checks whether they induce a partition of \( Y_K \) as observed for Lemma 6.4 (cf. Lines 7-13). If so, we have proved that \( M \) satisfies dependency fairness. If not, the analysis returns a set \( B \) of abstract-domain elements over-approximating the input regions in which bias might occur.

**Theorem 7.1.** If \( \text{ANALYZE}(M, K, Y, A) \) of Algorithm 1 returns \( \text{TRUE} \), \( \emptyset \) then \( M \) satisfies \( \mathcal{F}_K[Y] \).
Algorithm 1: A Naïve Backward Analysis

1: function BACKWARD(M, A, x)  
2:    a ← OUTCOME[|x|](NEW_A)  
3:    for i ← N − 1 down to 0 do  
4:       for j ← |L_i| down to 0 do  
5:          a ← ASSIGN(\text{ReLU}_A[|x_{i,j}|])  
6:    return a  
7: function CHECK(O)  
8:    B ← ∅  
9:    for all o_1, a_1 ∈ O do  
10:       for all o_2 ≠ o_1, a_2 ∈ O do  
11:          if a_1 \cap A_2 ≠ ⊥ then  
12:             B ← B ∪ {a_1 \cap A_2, a_2}  
13:    return B  
14: function ANALYZE(M, K, Y, A)  
15:    O ← ∅  
16:    for j ← 0 up to |L_N| do  
17:       a ← BACKWARD(M, A, x_{n,j})  
18:       O ← O ∪ \{x_{n,j} ↦ (\text{ASSUME}_A[|Y|](a)|_K)\}  
19:    B ← CHECK(O)  
20:    return B = ∅, B  

Proof (Sketch). ANALYZE(M, K, Y, A) in Algorithm 1 computes an over-approximation a of the regions of the input space that yield each target class x_{n,j} (cf. Line 17). Thus, it actually computes an over-approximation \{M\}_{n,j}^a of the dependency semantics \{M\}_{n,j}^M, i.e., \{M\}_{n,j}^M \supseteq \{M\}_{n,j}^a. Thus, if \{M\}_{n,j}^M satisfies \mathcal{F}_K[Y], i.e., ∀A, B ∈ \{M\}_{n,j}^M: (A_o ≠ B_o ⇒ A_o|_K \cap B_o|_K = ∅) (according to Lemma 6.4, cf. Line 19), then by transitivity we can conclude that also \{M\}_{n,j}^a necessarily satisfies \mathcal{F}_K[Y].  

In the analysis implementation, there is a tradeoff between performance and precision, which is reflected in the choice of abstract domain A and its transfer functions. Unfortunately, existing numerical abstract domains that are less expressive than polyhedra [Cousot and Halbwachs 1978] would make for a rather fast but too imprecise analysis. This is because they are not able to precisely handle constraints like \max X_N = x, which are introduced by OUTCOME_A[|x|] to partition with respect to outcome.

Furthermore, even polyhedra would not be precise enough in general. Indeed, each \text{ReLU}_A[|x_{i,j}|] would over-approximate what effectively is a conditional branch. Let |M| \defeq |L_1| + \cdots + |L_{N−1}| denote the number of hidden nodes (i.e., the number of ReLUs) in a model M. On the other side of the spectrum, one could use a disjunctive completion [Cousot and Cousot 1979] of polyhedra, thus keeping a separate polyhedron for each branch of a ReLU. This would yield a precise (in fact, exact) but extremely slow analysis: even with parallelization (cf. Line 16), each of the |L_N| processes would have to effectively explore 2^{|M|} paths!

In the rest of the paper, we improve on this naïve analysis and show how far we can go all the while remaining exact by using disjunctive polyhedra.
8 PARALLEL SEMANTICS

We first have to take a step back and return to reasoning at the concrete-semantics level. At the end of Section 6, we observed that the dependency semantics of a neural-network model \( M \) satisfying \( \mathcal{F}_k[Y] \) effectively induces a partition of \( Y \). We call this input partition fair.

More formally, given a set \( Y \) of initial states of interest, we say that an input partition \( \Pi \) of \( Y \) is fair if all value choices \( \forall x \) for the sensitive input nodes \( K \) of \( M \) are possible in all elements of the partitions: \( \forall I \in \Pi, Y \in \forall x \exists I : s(K) = Y \). For instance, \( \Pi = \{ T, T' \} \), with \( T \) and \( T' \) in Example 5.1 is a fair input partition of \( Y = \{ s \mid s(x_0, 1) = 0.5 \lor s(x_0, 1) = 0.75 \} \).

Given a fair input partition \( \Pi \) of \( Y \), the following result shows that we can verify whether a model \( M \) satisfies \( \mathcal{F}_k[Y] \) for each element \( I \) of \( \Pi \), independently.

**Lemma 8.1.** \( M \models \mathcal{F}_k[Y] \iff \forall I \in \Pi : \forall A, B \in \langle M \rangle_\Pi : (A_{\omega} \neq B_{\omega} \Rightarrow A_{|R_k} \cap B_{|R_k} = \emptyset) \)

**Proof.** The proof follows trivially from Lemma 6.4 and the fact that \( \Pi \) is a fair partition. \( \square \)

We use this new insight to further abstract the dependency semantics \( \Lambda_{\omega} \). We have the following Galois connection

\[
\langle P (P (\Sigma \times \Sigma)) \rangle_{\alpha_I} \leftrightarrow \langle P (P (\Sigma \times \Sigma)) \rangle_{\alpha_I},
\]

where \( \alpha_I(S) \equiv \{ R^I \mid R \in S \land I \in \Pi \} \). Here, the order \( \alpha_I \) is the pointwise ordering between sets of pairs of states restricted to first elements in the same \( I \in \Pi \), i.e., \( A \equiv \Pi B \equiv \bigwedge_{I \in \Pi} A^I \subseteq B^I \), where \( S^I \) denotes the only non-empty set of pairs in \( S^I \). We can now derive the parallel semantics \( \Pi_{\omega} \in P (P (\Sigma \times \Sigma)) \):

\[
\Pi_{\omega} \equiv \alpha_I(\Pi_{\omega}) = \{ \{ \sigma_0, \sigma_{\omega} \} \mid \sigma \in \Sigma \} \mid I \in \Pi \land O \in \Omega \}
\]

(11)

In fact, we derive a hierarchy of semantics, as depicted in Figure 2. We write \( \langle M \rangle_{\omega \omega} \) to denote the parallel semantics of a particular neural-network model \( M \). It remains to show soundness and completeness for \( \Pi_{\omega} \).

**Theorem 8.2.** \( M \models \mathcal{F}_k[Y] \iff \langle M \rangle_{\omega} \subseteq \alpha_I(\alpha_{\omega}(\mathcal{F}_k[Y])) \)

**Proof.** Let \( M \models \mathcal{F}_k[Y] \). From Theorem 6.2, we have that \( \langle M \rangle_{\omega} \subseteq \alpha_{\omega}(\alpha_{\omega}(\mathcal{F}_k[Y])) \). Thus, from the Galois connection in Equation 10, we have \( \alpha_I(\langle M \rangle_{\omega}) \subseteq \alpha_I(\alpha_{\omega}(\mathcal{F}_k[Y])) \). From the definition of \( \langle M \rangle_{\omega} \) (cf. Equation 11), we can then conclude that \( \langle M \rangle_{\omega} \subseteq \alpha_I(\alpha_{\omega}(\mathcal{F}_k[Y])) \). \( \square \)

**Corollary 8.3.** \( M \models \mathcal{F}_k[Y] \iff \langle M \rangle_{\omega} \subseteq \alpha_I(\alpha_{\omega}(\mathcal{F}_k[Y])) \)

Fig. 2. Hierarchy of semantics.
PROOF. The proof follows trivially from the definition of $\mathcal{S}$ (cf. Equation 6, 8, and 10) and Lemma 6.1 and 8.1.

Finally, from Lemma 8.1, we have that we can equivalently verify whether $\{M\}_A^\mathcal{l} \subseteq \mathcal{S}(\mathcal{P}_K[Y])$ by checking if the parallel semantics $\{M\}_A^\mathcal{l}$ induces a partition of each $I_K$.

Lemma 8.4. $M \models \mathcal{F}_K[Y] \iff \forall I \in \mathcal{I}: \forall A, B \in \{M\}_A^\mathcal{l}: (A^I \neq B^I \Rightarrow A^I_{0|K} \cap B^I_{0|K} = \emptyset)$

PROOF. The proof follows trivially from Lemma 8.1.

9 PARALLEL DEPENDENCY-FAIRNESS ANALYSIS

In this section, we build on the parallel semantics to design our novel perfectly parallel static analysis for dependency fairness, which automatically finds a fair partition $I$ and computes a sound over-approximation $\Pi_{a\alpha}$ of $\Pi_{\alpha\alpha}$, i.e., $\Pi_{\alpha\alpha} \subseteq \Pi_{a\alpha}$.

ReLU Activation Functions. We again only consider ReLU activation functions for now and postpone the discussion of other activation functions to the end of the section. The analysis is described in Algorithm 2. It combines a forward pre-analysis (Lines 15-24) with a backward analysis (Lines 28-38). The forward pre-analysis uses an abstract domain $A_1$ and builds partition $I$, while the backward analysis uses an abstract domain $A_2$ and performs the actual dependency-fairness analysis of a neural-network model $M$ with respect to its sensitive input nodes $K$ and a (representation of a) set of initial states $Y$ (cf. Line 13).

More specifically, the forward pre-analysis bounds the number of paths that the backward analysis has to explore. Indeed, not all of the $2^{|M|}$ paths of a model $M$ are necessarily viable starting from its input space.

In the rest of this section, we represent each path by an activation pattern, which determines the activation status of every ReLU operation in $M$. More precisely, an activation pattern is a sequence of flags. Each flag $p_{i,j}$ represents the activation status of the ReLU operation used to compute the value of hidden node $x_{i,j}$. If $p_{i,j}$ is $(i,j)$, the ReLU is always active, otherwise the ReLU is always inactive and $p_{i,j}$ is $x^{-i,j}$.

An abstract activation pattern gives the activation status of only a subset of the ReLUs of $M$, and thus, represents a set of activation patterns. ReLUs whose corresponding flag does not appear in an abstract activation pattern have an unknown (i.e., not fixed) activation status. Typically, only a relatively small number of abstract activation patterns is sufficient for covering the entire input space of a neural-network model. The design of our analysis builds on this key observation.

We set an analysis budget by providing an upper bound $U$ (cf. Line 13) on the number of tolerated ReLUs with an unknown activation status for each element $I$ of $\mathcal{I}$, i.e., on the number of paths that are to be explored by the backward analysis in each $I$. The forward pre-analysis starts with the trivial partition $I = \{Y\}$ (cf. Line 15). It proceeds forward for each element $I$ in $\mathcal{I}$ (cf. Lines 17-18). The transfer function $\mathcal{ReLU}\alpha_{\beta}[X_{i,j}]$ considers a ReLU operation and additionally builds an abstract activation pattern $p$ for $I$ (cf. Line 5) starting from the empty pattern $\varepsilon$ (cf. Line 2).

If $I$ leads to a unique outcome (cf. Line 19), then dependency fairness is already proved for $I$, and there is no need for a backward analysis; $I$ is added to the set of completed partitions (cf. Line 20). Instead, if abstract activation pattern $p$ fixes the activation status of enough ReLUs (cf. Line 21), we say that the backward analysis for $I$ is feasible. In this case, the pair of $p$ and $I$ is inserted into a map $F$ from abstract activation patterns to feasible partitions (cf. Line 22). The insertion takes care of merging abstract activation patterns that are subsumed by other (more) abstract patterns. In other words, it groups partitions whose abstract activation patterns fix more ReLUs with partitions whose patterns fix fewer ReLUs, and therefore, represent a superset of (concrete) patterns.
Algorithm 2: Our Analysis Based on Activation Patterns

1: function forward(M, A, I)
2:     a, p ← assume_A[I](new_A), ε
3:     for i ← 1 up to n do
4:         for j ← 0 up to |L_i| do
6:     return a, p
7: function backward(M, A, O, p)
8:     a ← outcome_A[O](new_A)
9:     for i ← n − 1 down to 0 do
10:        for j ← |l_i| down to 0 do
12:     return a
13: function analyze(M, K, Y, A_1, A_2, L, U)
14:     F, E, C ← ∅, ∅, ∅▷ F: feasible, E: excluded, C: completed
15:     I ← {Y}▷ perfectly parallelizable
16:     while I ≠ ∅ do
17:         a, p ← forward(M, A_1, I)
18:         if uniquely-classified(a) then
19:             C ← C ∪ {I}▷ I is already fair
20:         else if |M| − |p| ≤ U then
21:             F ← F ∪ {p ↦ I}▷ I is feasible
22:         else if |I| ≤ L then
23:             E ← E ∪ {p ↦ I}▷ I is excluded
24:         else
25:             I ← I ∪ partition_K(I)▷ I must be partitioned further
26:             B ← ∅▷ B: biased
27:     for all p, I ∈ F do
28:         O ← ∅▷ perfectly parallelizable
29:         for j ← 0 up to |L_n| do
30:             a ← backward(M, A_2, x_{n,j}, p)
31:             O ← O ∪ {x_{n,j} ↦ a}
32:     for all I ∈ I do
33:         O’ ← ∅
34:         for all o, a ∈ O do
35:             O’ ← O’ ∪ {o ↦ (assume_{A_2}[I]a)_{K}]
36:     B ← B ∪ check(O’)
37:     C ← C ∪ {I}
38:     return C, B = ∅, B, E▷ fair: B = ∅, maybe biased: B ≠ ∅

Otherwise, I needs to be partitioned further, with respect to K (cf. Line 25). Partitioning may continue until the size of I is smaller than the given lower bound L (cf. Lines 13 and 23). At this
point, I is set aside and excluded from the analysis until more resources (a larger upper bound U or a smaller lower bound L) become available (cf. Line 24).

Note that the forward pre-analysis lends itself to choosing a relatively cheap abstract domain $A_1$ since it does not need to precisely handle polyhedral constraints (like $\max X_N = x$, needed to partition with respect to outcome, cf. Section 7).

The analysis then proceeds backwards, independently for each abstract activation path $p$ and associated group of partitions $I$ (cf. Lines 28 and 31). The transfer function $\text{ReLU}_A^p [x_{i,j}]$ uses $p$ to choose which path(s) to explore at each ReLU operation, i.e., only the active (resp. inactive) path if $x_{i,j}$ (resp. $\bar{x}_{i,j}$) appears in $p$, or both if the activation status of the ReLU corresponding to hidden node $x_{i,j}$ is unknown. The (as we have seen, necessarily) expensive backward analysis only needs to run for each abstract activation pattern in the feasible map $F$. This is also why it is advantageous to merge subsumed abstract activation paths as described above.

Finally, the analysis checks dependency fairness of each element $I$ associated to $p$ (cf. Line 37). The analysis returns the set of input-space regions $C$ that have been completed and a set $B$ of abstract-domain elements over-approximating the regions in which bias might occur (cf. Line 39). If $B$ is empty, then the given model $M$ satisfies dependency fairness with respect to $K$ and $Y$ over $C$.

**Theorem 9.1.** If function $\text{analyze}(M, K, Y, A_1, A_2, L, U)$ in Algorithm 2 returns $C, \text{true}, \emptyset$, then $M$ satisfies $\mathcal{F}_K[Y]$ over the input-space fraction $C$.

**Proof (Sketch).** $\text{analyze}(M, K, Y, A_1, A_2, L, U)$ in Algorithm 2 first computes the abstract activation patterns that cover a fraction $C$ of the input space in which the analysis is feasible (Lines 15-24). Then, it computes an over-approximation $\mathcal{A}$ of the regions of $C$ that yield each target class $x_{m,j}$ (cf. Line 31). Thus, it actually computes an over-approximation $\llbracket M \rrbracket^{\mathcal{A}}_{\omega}$ of the parallel semantics $\llbracket M \rrbracket^{\omega}_{\omega}$, i.e., $\llbracket M \rrbracket^{\mathcal{A}}_{\omega} \supseteq \llbracket M \rrbracket^{\omega}_{\omega}$. Thus, if $\llbracket M \rrbracket^{\mathcal{A}}_{\omega}$ satisfies $\mathcal{F}_K[Y]$, i.e., $\forall I \in \mathcal{I}; \forall A, B \in \llbracket M \rrbracket^{\mathcal{A}}_{\omega}$: $(A^1_{\omega} \neq B^1_{\omega} \Rightarrow A^0_{\mathcal{R}} \cap B^0_{\mathcal{R}} = \emptyset)$ (according to Lemma 8.4, cf. Lines 33-37), then by transitivity we can conclude that also $\llbracket M \rrbracket^{\mathcal{A}}_{\omega}$ necessarily satisfies $\mathcal{F}_K[Y]$.

**Remark.** Recall that we assumed neural-network nodes to have real values (cf. Section 4). Thus, Theorem 9.1 is true for all choices of classical numerical abstract domains [Cousot and Cousot 1976; Cousot and Halbwachs 1978; Ghorbal et al. 2009; Miné 2006b, etc.] for $A_1$ and $A_2$. If we were to consider floating-point values instead, the only sound choices would be floating-point abstract domains [Chen et al. 2008; Miné 2004; Singh et al. 2019].

**Other Activation Functions.** Let us discuss how activation functions other than ReLUs would be handled. The only difference in Algorithm 2 would be the transfer functions $\text{ReLU}_A^p [x_{i,j}]$ (cf. Line 5) and $\text{ReLU}_A^p [x_{i,j}]$ (cf. Line 11), which would have to be replaced with the transfer functions corresponding to the considered activation function.

Piecewise-linear activation functions, like Leaky ReLU$(x) = \max(x, k \cdot x)$ or Hard TanH$(x) = \max(-1, \min(x, 1))$, can be treated analogously to ReLUs. The case of Leaky ReLUs is trivial. For Hard TanHs, the patterns $p$ used in Algorithm 2 will consist of flags $p_{i,j}$ with three possible values, depending on whether the corresponding hidden node $x_{i,j}$ has value less than or equal to $-1$, greater than or equal to $1$, or between $-1$ and $1$. For these activation functions, our approach remains sound and, in practice, exact when using disjunctive polyhedra for the backward analysis.

Other activation functions, e.g., Sigmoid$(x) = \frac{1}{1 + e^{-x}}$, can be soundly over-approximated [Singh et al. 2019] and similarly treated in a piecewise manner. In this case, however, we necessarily lose the exactness of the analysis, even when using disjunctive polyhedra.
10 IMPLEMENTATION

We implemented our dependency-fairness analysis described in the previous section in a tool called Libra. The implementation is written in PYTHON and is open source\(^4\).

**Tool Inputs.** Libra takes as input a neural-network model \(M\) expressed as a PYTHON program (cf. Section 3), a specification of the input layer \(l_0\) of \(M\), an abstract domain for the forward pre-analysis, and budget constraints \(L\) and \(U\). The specification for \(l_0\) determines which input nodes correspond to continuous and (one-hot encoded) categorical data and, among them, which should be considered bias sensitive. We assume that continuous data is in the range \([0, 1]\). A set \(Y\) of initial states of interest is specified using an assumption at the beginning of the program representation of \(M\).

**Abstract Domains.** For the forward pre-analysis, choices of the abstract domain are either boxes \([\text{Cousot and Cousot 1976}]\) (i.e., boxes in the following), or a combination of boxes and symbolic constant propagation \([\text{Li et al. 2019}; \text{Miné 2006a}]\) (i.e., symbolic in the following), or the deeppoly domain \([\text{Singh et al. 2019}]\), which is designed for proving local robustness of neural networks. As previously mentioned, we use disjunctive polyhedra for the backward analysis. All abstract domains are built on top of the apron abstract-domain library \([\text{Jeannet and Miné 2009}]\).

**Parallelization.** Both the forward and backward analyses are parallelized to run on multiple CPU cores. The pre-analysis uses a queue from which each process draws a fraction \(I\) of \(Y\) (cf. Line 17). Fractions that need to be partitioned further are split in half along one of the non-sensitive dimensions (in a round-robin fashion), and the resulting (sub)fractions are put back into the queue (cf. Line 26). Feasible \(I_s\) (with their corresponding abstract activation pattern \(p\)) are put into another queue (cf. Line 22) for the backward analysis.

**Tool Outputs.** The analysis returns the fractions of \(Y\) that were analyzed and any (sub)regions of these where bias was found. It also reports the percentage of the input space that was analyzed and (an estimate of) the percentage that was found biased according to a given probability distribution of the input space (uniform by default). To obtain the latter, we simply use the size of a box wrapped around each biased region. More precise but also costlier solutions exist \([\text{Barvinok 1994}]\).

In general, how much of \(Y\) can be analyzed depends on the analysis configuration (i.e., chosen abstract domain for the forward pre-analysis and budget constraints \(L\) and \(U\)). For the fractions of \(Y\) that remained excluded from the analysis one can always re-run the analysis with a more powerful configuration, i.e., by restricting \(Y\) to the excluded fraction and choosing a more precise abstract domain or a higher \(L\) or a lower \(U\). We plan on automating this process as part of our future work.

11 EXPERIMENTAL EVALUATION

In this section, we evaluate our approach by focusing on the following research questions:

- **RQ1:** Can our analysis detect seeded (i.e., injected) bias?
- **RQ2:** Is our analysis able to answer specific bias queries?
- **RQ3:** How does the model structure affect the scalability of the analysis?
- **RQ4:** How does the analyzed input-space size affect the scalability of the analysis?
- **RQ5:** How does the analysis budget affect the scalability-vs-precision tradeoff?
- **RQ6:** Can our analysis effectively leverage multiple CPUs?

\(^4\)https://github.com/caterinaurban/Libra
Perfectly Parallel Fairness Certification of Neural Networks

11.1 Data

For our evaluation, we used public datasets from the UCI Machine Learning Repository and ProPublica (see below for more details) to train several neural-network models. We primarily focused on datasets discussed in the literature [Mehrabi et al. 2019] or used by related techniques (e.g., [Albarghouthi et al. 2017a,b; Albarghouthi and Vinitsky 2019; Bastani et al. 2019; Datta et al. 2017; Galhotra et al. 2017; Tramèr et al. 2017; Udeshi et al. 2018]) and recent work on fairness targeting feed-forward neural networks (e.g., [Manisha and Gujar 2020; Yurochkin et al. 2020]).

We pre-processed these datasets both to make them fair with respect to a certain sensitive input feature as well as to seed bias. This way, we eliminate as many sources of uncontrolled bias as possible, i.e., bias originally present in the dataset as well as bias potentially introduced by the training process due to under-representation in the dataset (as discussed in the Introduction). We describe how we seeded bias in each particular dataset later on in this section.

Our methodology for making the data fair was common across datasets. Specifically, given an original dataset and a sensitive feature (say, race), we selected the largest population with a particular value for this feature (say, Caucasian) from the dataset (and discarded all others). We removed any duplicate or inconsistent entries from this population. We then duplicated the population for every other value of the sensitive feature (say, Asian and Hispanic). For example, assuming the largest population was 500 Caucasians, we created 500 Asians and 500 Hispanics, and any two of these populations differ only in the value of race. Consequently, the new dataset is fair because there do not exist two inputs $k$ and $k'$ that differ only in the value of the sensitive feature for which the classification outcomes are different. A similar methodology is used by Yurochkin et al. in recent related work [Yurochkin et al. 2020].

We define the unfairness score of a dataset as the percentage of inputs $k$ in the dataset for which there exists another input $k'$ that differs from $k$ only in the value of the sensitive feature and the classification outcome. Our fair datasets have an unfairness score of 0%.

All datasets used in our experiments are open source as part of LIBRA.

11.2 Setup

Since neural-network training is non-deterministic, we typically train eight neural networks on each dataset, unless stated otherwise. The model sizes range from 2 hidden layers with 5 nodes each to 32 hidden layers with 40 nodes each. All models were trained with Keras, using the RMSprop optimizer with the default learning rate, and categorical crossentropy as the loss function. Each model was trained for 50 iterations. With these settings, we generally obtain lower accuracy values than those reported in the literature (e.g., [Manisha and Gujar 2020]). We believe that this is largely due to the fact that we modified the original datasets to make them fair or to seed bias and, more importantly, the fact that we did not invest into hyperparameter optimizations. However, we remark and demonstrate below that training neural networks with higher accuracy is not needed to show that our analysis works (i.e., is indeed able to certify fairness or detect bias) and to evaluate its performance. All models used in our experiments are open source as part of LIBRA. For each model, we assume a uniform distribution of the input space.

We performed all experiments on a 12-core Intel ® Xeon ® X5650 CPU @ 2.67GHz machine with 48GB of memory, running Debian GNU/Linux 9.6 (stretch).

11.3 Results

In the following, we present our experimental results for each of the above research questions.

**RQ1: Detecting Seeded Bias.** This research question focuses on detecting seeded bias by comparing the analysis results for models trained with fair versus biased data.
We seeded bias in the fair dataset by randomly assigning a bad credit score to people of age 60 and above who request a credit amount of more than EUR 1 000 until we reached a 20% unfairness score of the dataset. The median classification accuracy of the models (17 inputs and 4 hidden layers with 5 nodes each) trained on fair and biased data was 71% and 65%, respectively.

To analyze these models, we set $L = 0$ to be sure to complete the analysis on 100% of the input space. The drawback with this is that the pre-analysis might end up splitting input partitions endlessly. To counteract, for each model, we chose the smallest upper bound $U$ that did not cause this issue (i.e., we used values for $U$ between 1 and 16). Table 1 shows the analysis results for the different choices of domain used for the forward pre-analysis. In particular, it shows whether the models are biased with respect to age for credit requests of 1 000 or less as well as for credit requests of over 1 000. Columns $\texttt{bias}$ and $\texttt{time}$ show the detected bias (in percentage of the entire input space) and the analysis running time. We show minimum, median, and maximum bias percentage and running time for each credit request group. For each line in Table 1, we highlighted the choice of the abstract domain that entailed the shortest analysis time.

In this case, we expect the analysis to detect the highest bias percentage for credit requests of over 1 000 and for the models trained on biased data. This is indeed what we obtain: the analysis finds 3.5× median bias for high credit amounts compared to models trained on fair data. This demonstrates that our approach is able to effectively detect seeded bias.

For models trained on fair data, we observe a maybe unexpected difference in the bias found for small credit amounts compared to large credit amounts. This is in part due to the fact that bias is given in percentage of the entire input space and not scaled with respect to the analyzed input space (small credit amounts correspond to a mere 4% of the input space). When scaling the bias percentage with respect to the analyzed input space, the difference is less marked: the median bias is 0.19% / 4% = 4.75% for small credit amounts and 6.72% / 96% = 7% (or 6.63% / 96% = 6.9% for the $\texttt{deeppoly}$ domain) for large credit amounts. The remaining difference indicates that the models contain bias that does not necessarily depend on the credit amount. The bias is introduced by the training process itself (as explained in the Introduction) and is not due to imprecision of our analysis. Recall that our approach is exact, and imprecision is only introduced when estimating the bias percentage (cf. Section 10). Similar considerations justify the difference in the bias found for small credit amounts for models trained on fair data compared to those trained on biased data.

Finally, for comparison, the analysis of models trained on the original dataset (with median accuracy of 74%) found 0.28% bias for small credit amounts and 17.7% bias for large credit amounts.

**RQ2: Answering Bias Queries.** To further evaluate the precision of our approach, we created queries concerning bias within specific groups of people, each corresponding to a subset of the

---

5[https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)]
Table 2. Queries on Models Trained on Fair and Race-Biased Data (ProPublica’s COMPAS Data)

<table>
<thead>
<tr>
<th>QUERY</th>
<th>BOXES</th>
<th>SYMBOLIC</th>
<th>DEEPPOLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAIR DATA</td>
<td>BIASED DATA</td>
<td>FAIR DATA</td>
<td>BIASED DATA</td>
</tr>
<tr>
<td>AGE &lt; 25</td>
<td>0.22%</td>
<td>0.12%</td>
<td>0.22%</td>
</tr>
<tr>
<td>RACE BIAS?</td>
<td>0.31%</td>
<td>0.31%</td>
<td>0.32%</td>
</tr>
<tr>
<td>CAUCASIAN PRIORS BIAS?</td>
<td>5.36%</td>
<td>6.98%</td>
<td>5.36%</td>
</tr>
<tr>
<td>MALE</td>
<td>2.60%</td>
<td>2.51%</td>
<td>2.60%</td>
</tr>
<tr>
<td>AGE BIAS?</td>
<td>6.08%</td>
<td>6.93%</td>
<td>6.08%</td>
</tr>
<tr>
<td>COCAIN PRIORS BIAS?</td>
<td>2.95%</td>
<td>4.21%</td>
<td>2.95%</td>
</tr>
</tbody>
</table>

entire input space. We used the COMPAS dataset from ProPublica for this experiment. The data assigns a three-valued recidivism-risk score (high, medium, and low) indicating how likely criminals are to re-offend. The data includes both personal attributes (e.g., age and race) as well as criminal history (e.g., number of priors and violent crimes). As for RQ1, we trained models both on fair and biased data. Here, we considered race as the sensitive feature. We seeded bias in the fair data by randomly assigning high recidivism risk to African Americans until we reached a 20% unfairness score of the dataset. The median classification accuracy of the 3-class models (19 inputs and 4 hidden layers with 5 nodes each) trained on fair and biased data was 55% and 56%, respectively.

To analyze these models, we used a lower bound L of 0, and an upper bound U between 7 and 19. Table 2 shows the results of our analysis (i.e., columns shown as in Table 1) for three queries, each representing a different choice for the set K of sensitive features:

- **Q_A:** Is there bias with respect to race for people younger than 25?
- **Q_B:** Is there bias with respect to age for males?
- **Q_C:** Is there bias with respect to the number of priors for Caucasians?

In this case, we expect the result of the analysis to change coherently according to the choice of K. Specifically, Q_A is expected to show a significant difference in the bias detected for models trained on fair data compared to models trained on biased data, while Q_B and Q_C should not show marked differences between fair and biased models. However, bias with respect to number of priors seems natural in the context of the dataset (i.e., recidivism risk increases with higher number of priors) and, therefore, we expect the bias percentage obtained for Q_C to be higher than that found for Q_B across both sets of fair and biased neural-network models.

The results in Table 2 meet our expectations. For Q_A, the analysis detects about three times as much median bias for the models trained on biased data compared to those trained on fair data. In contrast, for Q_B, the analysis finds a comparable amount of age bias across both sets of models. This becomes more evident when scaling the median bias with respect to the queried input space (males correspond to 50% of the input space): the smallest median bias for the models trained on fair data is 12.16% (for the boxes domain) and the largest median bias for the models trained on biased data is 14.04% (for the symbolic domain). Finally, for Q_C, the analysis detects significant bias across both sets of models with respect to the number of priors. When considering the queried input space (Caucasians represent 1/6 of the entire input space), this translates to 17.7% median bias for the models trained on fair data and 25.26% for the models trained on biased data. Overall, these results demonstrate the effectiveness of our analysis in answering specific bias queries. For comparison, the analysis of neural-network models trained on the original dataset (with median accuracy of 63%) found 1.21% median bias for Q_A, 5.34% median bias for Q_B, and 5.86% median bias for Q_C.

---


Table 3. Comparison of Different Model Structures (Adult Census Data)

<table>
<thead>
<tr>
<th>M</th>
<th>U</th>
<th>INPUT</th>
<th>C</th>
<th>BOXES</th>
<th>SYMBOLIC</th>
<th>DEEPPOLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>6</td>
<td>99.51%</td>
<td>769</td>
<td>51</td>
<td>723</td>
<td>1h 16m 25s</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>100.00%</td>
<td>152</td>
<td>19</td>
<td>143</td>
<td>3h 47m 23s</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>49.83%</td>
<td>719</td>
<td>9</td>
<td>329</td>
<td>13m 43s</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>58.68%</td>
<td>642</td>
<td>9</td>
<td>284</td>
<td>1h 40m 43s</td>
</tr>
<tr>
<td>10</td>
<td>99.96%</td>
<td>313</td>
<td>7</td>
<td>260</td>
<td>1h 21m 47s</td>
<td>99.96%</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>46.22%</td>
<td>1123</td>
<td>62</td>
<td>255</td>
<td>20m 51s</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>64.24%</td>
<td>1111</td>
<td>96</td>
<td>782</td>
<td>2h 24m 51s</td>
</tr>
<tr>
<td>40</td>
<td>6</td>
<td>85.90%</td>
<td>1390</td>
<td>74</td>
<td>1339</td>
<td>&gt;13h</td>
</tr>
<tr>
<td>40</td>
<td>8</td>
<td>0.35%</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1m 39s</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>0.42%</td>
<td>12</td>
<td>1</td>
<td>2</td>
<td>14m 37s</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>0.80%</td>
<td>21</td>
<td>10</td>
<td>13</td>
<td>1h 48m 49s</td>
</tr>
<tr>
<td>45</td>
<td>6</td>
<td>2.50%</td>
<td>72</td>
<td>3</td>
<td>22</td>
<td>4m 35s</td>
</tr>
<tr>
<td>45</td>
<td>8</td>
<td>9.83%</td>
<td>282</td>
<td>25</td>
<td>234</td>
<td>2h 25m 38s</td>
</tr>
<tr>
<td>10</td>
<td>18.66%</td>
<td>522</td>
<td>13</td>
<td>488</td>
<td>1h 51m 12s</td>
<td>49.62%</td>
</tr>
</tbody>
</table>

Fig. 3. Comparison of Different Model Structures (Adult Census Data)

For each line in Table 2, we highlighted the choice of abstract domain that entailed the shortest analysis time. We observe that DEEPPOLY seems generally the better choice. The difference in performance becomes more striking as the analyzed input space becomes smaller, i.e., for QC. This is because DEEPPOLY is specifically designed for proving local robustness of neural networks. Thus, our input partitioning, in addition to allowing for parallelism, is also enabling analyses designed for local properties to prove global properties, like dependency fairness.

**RQ3: Effect of Model Structure on Scalability.** To evaluate the effect of the model structure on the scalability of our analysis, we trained models on the Adult Census dataset\(^7\) by varying the number of layers and nodes per layer. The dataset assigns a yearly income (> or \(\leq\) USD 50K) based on personal attributes such as gender, race, and occupation. All models (with 23 inputs) were trained on a fair dataset with respect to gender and had minimum classification accuracy of 78%.

Table 3 shows the results. The first column (\(|M|\)) shows the total number of hidden nodes and introduces the marker symbols used in the scatter plot of Figure 3 (to identify the domain used for

\(^7\)https://archive.ics.uci.edu/ml/datasets/adult
the forward pre-analysis: left, center, and right symbols respectively refer to the BOXES, SYMBOLIC, and DEEPPOLY domains). The models have the following number of hidden layers and nodes per layer (from top to bottom): 2 and 5; 4 and 3; 4 and 5; 4 and 10; 9 and 5.

Column U shows the chosen upper bound for the analysis. For each model, we tried four different choices of U. Column input shows the input-space coverage, i.e., the percentage of the input space that was completed by the analysis. Column |C| shows the total number of analyzed (i.e., completed) input space partitions. Column |F| shows the total number of abstract activation patterns (left) and feasible input partitions (right) that the backward analysis had to explore. The difference between |C| and the number of partitions shown in |F| are the input partitions that the pre-analysis found to be already fair (i.e., uniquely classified). Finally, column time shows the analysis running time. We used a lower bound L of 0.5 and a time limit of 13h. For each model in Table 3, we highlighted the configuration (i.e., domain used for the pre-analysis and chosen U) that achieved the highest input-space coverage (the analysis running time being decisive in case of equality or timeout).

The scatter plot of Figure 3a visualizes the input coverage and analysis running time. We zoom in on the best U-configurations for each pre-analysis domain (i.e., the chosen U) in Figure 3b.

Overall, we observe that coverage decreases for larger model structures, and the more precise SYMBOLIC and DEEPPOLY domains result in a significant coverage boost, especially for larger structures. We also note that, as in this case we are analyzing the entire input space, DEEPPOLY generally performs worse than the SYMBOLIC domain. In particular, for larger structures, the SYMBOLIC domain often yields a higher input coverage in a shorter analysis running time. Interestingly, the input coverage is higher for the largest model with 45 hidden nodes (i.e., 9 hidden layers with 5 nodes each) than that with 40 hidden nodes (i.e., 4 hidden layers with 10 nodes each). In fact, in neural-network models with more layers but fewer nodes per layer, the activation status of a node fixes the activation status of more nodes in subsequent layers. As a consequence, fewer activation patterns are actually needed to cover a larger portion of the input space. For instance with the SYMBOLIC domain only 14 patterns are enough to cover 41.98% of the input space for the model with 45 hidden nodes with $U = 4$, while for the model with 40 hidden nodes already 50 patterns are needed to cover only 37.19% of the input space with $U = 10$. Finally, we observe that increasing the upper bound $U$ tends to increase coverage independently of the specific model structure. However, interestingly, this does not always come at the expense of an increased running time. In fact, such a change often results in decreasing the number of partitions that the expensive backward analysis needs to analyze (cf. columns |F|) and, in turn, this reduces the overall running time.

**RQ4: Effect of Analyzed Input Space on Scalability.** As said above, the analysis of the models considered in Table 3 is conducted on the entire input space. In practice, as already mentioned, one might be interested in just a portion of the input space, e.g., depending on the probability distribution. More generally, we argue that the size of the analyzed input space (rather than the size of the analyzed neural network) is the most important factor that affects the performance of the analysis. To support this claim, we trained even larger models and analyzed them with respect to queries exercising different input space sizes. Table 4 shows the results. The first column again shows the total number of hidden nodes for each trained model. In particular, the models we analyzed have the following number of hidden layers and nodes per layer (from top to bottom): 4 and 5; 8 and 10; 16 and 20; 32 and 40. Column query shows the query used for the analysis and the corresponding queried input space size. Specifically, the queries identify people with the following characteristics:

\[ A: \text{true} \]

queried input space: 100.0%
Table 4. Comparison of Different Input Space Sizes and Model Structures (Adult Census Data)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>F 0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
</tr>
<tr>
<td>80</td>
<td>F 0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
</tr>
<tr>
<td>320</td>
<td>F 0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
</tr>
<tr>
<td>1280</td>
<td>F 0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
<td>0.09%</td>
</tr>
</tbody>
</table>

\(B: A \land \text{age} \leq 53.5\)

\(C: B \land \text{race} = \text{white}\)

\(D: C \land \text{work class} = \text{private}\)

\(E: D \land \text{marital status} = \text{single}\)

\(F: E \land \text{occupation} = \text{blue-collar}\)

For the analysis budget, we used \(L = 0.25, U = 0.1 * [M]\), and a time limit of 13h. Column INPUT shows, for each domain used for the forward pre-analysis, the coverage of the queried input space (i.e., the percentage of the input space that satisfies the query and was completed by the analysis) and the corresponding input-space coverage (i.e., the same percentage but this time scaled to the entire input space). Columns \(U, [C], [F],\) and \(\text{TIME}\) are as before. Where a timeout is indicated (i.e., \(\text{TIME} > 13\)h) and the values for the INPUT, [C], and [F] columns are missing, it means that the timeout occurred during the pre-analysis; otherwise, it happened during the backward analysis. For each model and query, we highlighted the configuration (i.e., the abstract domain used for the

\(8\)This corresponds to \(\text{age} \leq 0.5\) with min-max scaling between 0 and 1.

Table 5. Comparison of Different Analysis Configurations (Japanese Credit Screening) — 12 CPUs

<table>
<thead>
<tr>
<th>L</th>
<th>U</th>
<th>INPUT</th>
<th>C</th>
<th>BOXES</th>
<th>TIME</th>
<th>SYMBOLIC</th>
<th>TIME</th>
<th>DEEPPOLY</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>15.28%</td>
<td>37</td>
<td>0</td>
<td>0</td>
<td>8s</td>
<td>58.33%</td>
<td>79</td>
<td>8</td>
</tr>
<tr>
<td>0.5</td>
<td>6</td>
<td>17.01%</td>
<td>39</td>
<td>6</td>
<td>6</td>
<td>51s</td>
<td>69.10%</td>
<td>120</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>51.39%</td>
<td>90</td>
<td>28</td>
<td>85</td>
<td>12m 26s</td>
<td>82.64%</td>
<td>88</td>
<td>31</td>
</tr>
<tr>
<td>10</td>
<td>79.86%</td>
<td>89</td>
<td>34</td>
<td>89</td>
<td>34m 15s</td>
<td>93.06%</td>
<td>98</td>
<td>40</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>59.09%</td>
<td>1115</td>
<td>20</td>
<td>415</td>
<td>54m 32s</td>
<td>95.94%</td>
<td>884</td>
<td>39</td>
</tr>
<tr>
<td>0.25</td>
<td>6</td>
<td>83.77%</td>
<td>1404</td>
<td>79</td>
<td>944</td>
<td>57m 19s</td>
<td>98.68%</td>
<td>634</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>96.07%</td>
<td>869</td>
<td>140</td>
<td>761</td>
<td>1h 7m 29s</td>
<td>99.72%</td>
<td>310</td>
<td>67</td>
</tr>
<tr>
<td>10</td>
<td>99.54%</td>
<td>409</td>
<td>93</td>
<td>403</td>
<td>1h 35m 20s</td>
<td>99.98%</td>
<td>195</td>
<td>52</td>
<td>176</td>
</tr>
<tr>
<td>0.125</td>
<td>4</td>
<td>97.13%</td>
<td>12449</td>
<td>200</td>
<td>9519</td>
<td>3h 33m 48s</td>
<td>99.99%</td>
<td>1101</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>99.83%</td>
<td>5919</td>
<td>276</td>
<td>4460</td>
<td>3h 23m</td>
<td>100.00%</td>
<td>986</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>99.98%</td>
<td>1926</td>
<td>203</td>
<td>1568</td>
<td>2h 14m 25s</td>
<td>100.00%</td>
<td>404</td>
<td>73</td>
</tr>
<tr>
<td>10</td>
<td>100.00%</td>
<td>428</td>
<td>95</td>
<td>427</td>
<td>1h 39m 31s</td>
<td>100.00%</td>
<td>151</td>
<td>53</td>
<td>141</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>100.00%</td>
<td>19299</td>
<td>295</td>
<td>15446</td>
<td>6h 13m 24s</td>
<td>100.00%</td>
<td>1397</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>100.00%</td>
<td>4843</td>
<td>280</td>
<td>3679</td>
<td>2h 24m 7s</td>
<td>100.00%</td>
<td>763</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>100.00%</td>
<td>1919</td>
<td>208</td>
<td>1567</td>
<td>2h 9m 5s</td>
<td>100.00%</td>
<td>404</td>
<td>73</td>
</tr>
<tr>
<td>10</td>
<td>100.00%</td>
<td>486</td>
<td>102</td>
<td>475</td>
<td>1h 41m 3s</td>
<td>100.00%</td>
<td>217</td>
<td>55</td>
<td>192</td>
</tr>
</tbody>
</table>

pre-analysis) that achieved the highest input-space coverage with the shortest analysis running time. Note that, where the [F] column only contains zeros, it means that the backward analysis had no activation patterns to explore; this implies that the entire covered input space (i.e., the percentage shown in the INPUT column) was already certified to be fair by the forward analysis.

Overall, we observe that whenever the analyzed input space is small enough (i.e., queries $D - F$), the size of the neural network has little influence on the input space coverage and slightly impacts the analysis running time, independently of the domain used for the forward pre-analysis. Instead, for larger analyzed input spaces (i.e., queries $A - C$) performance degrades quickly for larger neural networks. These results thus support our claim. In fact, these considerations generalize to other research areas in the verification of neural networks, e.g., in the certification of local robustness against adversarial examples: the size of the perturbation is the most important factor that affects the performance of the verification [Tran et al. 2020]. Finally, again, we observe that the SYMBOLIC domain generally is the better choice for the forward pre-analysis, in particular for queries exercising a larger input space or larger neural networks.

**RQ5: Scalability-vs-Precision Tradeoff.** To evaluate the effect of the analysis budget (bouds L and U), we analyzed a model using different budget configurations. For this experiment, we used the Japanese Credit Screening\(^9\) dataset, which we made fair with respect to gender. Our 2-class model (17 inputs and 4 hidden layers with 5 nodes each) had a classification accuracy of 86%.

Table 5 shows the results of the analysis for different budget configurations and choices for the domain used for the forward pre-analysis. The best configuration in terms of input-space coverage and analysis running time is highlighted. The symbol next to each domain name introduces the marker used in the scatter plot of Figure 4a, which visualizes the coverage and running time. Figure 4b zooms on 90.00% $\leq$ INPUT and 1000s $\leq$ TIME $\leq$ 1000s.

Overall, we observe that the more precise SYMBOLIC and DEEPPOLY domains boost input coverage, most noticeably for configurations with a larger L. This additional precision does not always result in longer running times. In fact, a more precise pre-analysis often reduces the overall running time. This is because the pre-analysis is able to prove that more partitions are already fair without requiring them to go through the backward analysis (cf. columns [F]).

Independently of the chosen domain for the forward pre-analysis, as expected, a larger U or a smaller L increase precision. Increasing U or L typically reduces the number of completed partitions.

\(^9\)https://archive.ics.uci.edu/ml/datasets/Japanese+Credit+Screening
generally increase the running time (when comparing configurations with similar coverage). Based on our experience, the optimal budget largely depends on the analyzed model.

**RQ6: Leveraging Multiple CPUs.** To evaluate the effect of parallelizing the analysis using multiple cores, we re-ran the analyses of RQ5 on 4 CPU cores instead of 12. Table 6 shows these results. We observe the most significant increase in running time for 4 cores for the BOXES domain. On average, the running time increases by a factor of 2.6. On the other hand, for the SYMBOLIC and DEEPPOLY domains, the running time with 4 cores increases less drastically, on average by a factor of 1.6 and 2, respectively. This is again explained by the increased precision of the forward analysis; fewer partitions require a backward pass, where parallelization is most effective.

Note that the differences between the columns \(|C|\) and \(|F|\) of Table 5 and Table 6 are due to random choices that happen during the analysis, i.e., mainly the order in which (continuous) non-sensitive features are listed, which in turn dictates the order in which the partitioning happens (cf. columns \(|C|\)). Consequently, partitions tend to be more complex, requiring both forward and backward analyses. Since the backward analysis tends to dominate the running time, more partitions generally increase the running time (when comparing configurations with similar coverage). Based on our experience, the optimal budget largely depends on the analyzed model.
during the analysis. As a consequence, in some cases the analysis finds fewer (resp. more) activation patterns to analyze and thus might complete a little faster (resp. slower). Finding a good criterion for identifying an optimal partitioning strategy for the analysis is left for future work.

Table 7 shows the results of the same experiment on 24 vCPUs.

12 RELATED WORK

Significant progress has been made on testing and verifying machine-learning models. We focus on fairness, safety, and robustness properties in the following, especially of deep neural networks.

Fairness Criteria. There are countless fairness definitions in the literature. In this paper, we focus on dependency fairness (originally called causal fairness [Galhotra et al. 2017]) and compare here with the most popular and related fairness notions.

Demographic parity [Feldman et al. 2015] (or group fairness) is the most common non-causal notion of fairness. It states that individuals with different values of sensitive features, hence belonging to different groups, should have the same probability of being predicted to the positive class. For example, a loan system satisfies group fairness with respect to gender if male and female applicants have equal probability of getting loans. If unsatisfied, this notion is also referred to as disparate impact. Our notion of fairness is stronger, as it imposes fairness on every pair of individuals that differ only in sensitive features. A classifier that satisfies group fairness does not necessarily satisfy dependency fairness, because there may still exist pairs of individuals on which the classifier exhibits bias.

Another group-based notion of fairness is equality of opportunity [Hardt et al. 2016]. It states that qualified individuals with different values of sensitive features should have equal probability of being predicted to the positive class. For a loan system, this means that male and female applicants who are qualified to receive loans should have an equal chance of being approved. By imposing fairness on every qualified pair of individuals that differ only in sensitive features, we can generalize dependency fairness to also concern both prediction and actual results. We can then adapt our technique to consider only the part of the input space that includes qualified individuals.

Other causal notions of fairness [Chiappa 2019; Kilbertus et al. 2017; Kusner et al. 2017; Nabi and Shpitser 2018, etc.] require additional knowledge in the form of a causal model [Pearl 2009]. A causal model can drive the choice of the sensitive input(s) for our analysis.
**Testing and Verifying Fairness.** Galhotra et al. [Galhotra et al. 2017] proposed an approach, Themis, that allows efficient fairness testing of software. Udeshi et al. [Udeshi et al. 2018] designed an automated and directed testing technique to generate discriminatory inputs for machine-learning models. Tramer et al. [Tramèr et al. 2017] introduced the unwarranted-associations framework and instantiated it in FairTest. In contrast, our technique provides formal fairness guarantees.

Bastani et al. [Bastani et al. 2019] used adaptive concentration inequalities to design a scalable sampling technique for providing probabilistic fairness guarantees for machine-learning models. As mentioned in the Introduction, our approach differs in that it gives definite (instead of probabilistic) guarantees. However, it might exclude partitions for which the analysis is not exact.

Albarghouthi et al. [Albarghouthi et al. 2017b] encoded fairness problems as probabilistic program properties and developed an SMT-based technique for verifying fairness of decision-making programs. As discussed in the Introduction, this technique has been shown to scale only up to neural networks with at most 3 inputs and a single hidden layer with at most 2 nodes. In contrast, our approach is designed to be perfectly parallel, and thus, is significantly more scalable and can analyze neural networks with hundreds (or, in some cases, even thousands) of hidden nodes.

A recent technique [Ruoss et al. 2020] certifies individual fairness of neural networks, which is a local property that coincides with robustness within a particular distance metric. In particular, individual fairness dictates that similar individuals should be treated similarly. Our approach, however, targets certification of neural networks for the global property of dependency fairness.

For certain biased decision-making programs, the program repair technique proposed by Albarghouthi et al. [Albarghouthi et al. 2017a] can be used to repair their bias. Albarghouthi and Vinitsky [Albarghouthi and Vinitsky 2019] further introduced fairness-aware programming, where programmers can specify fairness properties in their code for runtime checking.

**Robustness of Deep Neural Networks.** Robustness is a desirable property for traditional software [Chaudhuri et al. 2012; Goubault and Putot 2013; Majumdar and Saha 2009], especially control systems. Deep neural networks are also expected to be robust. However, research has shown that deep neural networks are not robust to small perturbations of their inputs [Szegedy et al. 2014] and can even be easily fooled [Nguyen et al. 2015]. Subtle imperceptible perturbations of inputs, known as adversarial examples, can change their prediction results. Various algorithms [Carlini and Wagner 2017b; Goodfellow et al. 2015; Madry et al. 2018; Tabacof and Valle 2016; Zhang et al. 2020] have been proposed that can effectively find adversarial examples. Research on developing defense mechanisms against adversarial examples [Athalye et al. 2018; Carlini and Wagner 2016, 2017a,b; Cornelius 2019; Engstrom et al. 2018; Goodfellow et al. 2015; Huang et al. 2015; Mirman et al. 2018, 2019] is also active. Dependency fairness is a special form of robustness in the sense that neural networks are expected to be globally robust with respect to their sensitive features.

**Testing Deep Learning Systems.** Multiple frameworks have been proposed to test the robustness of deep learning systems. Pei et al. [Pei et al. 2017] proposed the first whitebox framework for testing such systems. They used neuron coverage to measure the adequacy of test inputs. Sun et al. [Sun et al. 2018] presented the first concolic-testing [Godefroid et al. 2005; Sen et al. 2005] approach for neural networks. Tian et al. [Tian et al. 2018] and Zhang et al. [Zhang et al. 2018] proposed frameworks for testing autonomous driving systems. Gopinath et al. [Gopinath et al. 2018] used symbolic execution [Clarke 1976; King 1976]. Odena et al. [Odena et al. 2019] were the first to develop coverage-guided fuzzing for neural networks. Zhang et al. [Zhang et al. 2020] proposed a blackbox-fuzzing technique to test their robustness.

**Formal Verification of Deep Neural Networks.** Formal verification of deep neural networks has mainly focused on safety properties. However, the scalability of such techniques for verifying

13 CONCLUSION AND FUTURE WORK

We have presented a novel, automated, perfectly parallel static analysis for certifying fairness of feed-forward neural networks used for classification of tabular data — a problem of ever-increasing real-world importance nowadays. Our approach provides definite fairness guarantees for the analyzed input space and it is the first in this area that is configurable in terms of scalability and precision. The analysis can thus support a wide range of use cases throughout the development lifecycle of neural networks: ranging from quick sanity checks during development to formal fairness audits before deployments. We have rigorously formalized the approach and proved its soundness, and we demonstrated its effectiveness in practice with an extensive experimental evaluation.

In future work, we plan on integrating strategies in libra for making the parameter configuration of U and L automatic during the analysis. For instance, by starting the analysis with a low U (resp. large L) and gradually increasing (resp. decreasing) it only on the input space partitions on which it is necessary. We also plan on supporting other encodings for categorical features than one-hot encoding, such as entity embeddings [Guo and Berkhahn 2016], which have become more and more popular in recent years. Our approach will also automatically benefit from future advances in the design of abstract domains for analyzing neural networks. Vice versa, we believe and hope that future work can also build on our approach. For instance, other tools could feed on the results of our analysis and, say, provide weaker fairness guarantees (e.g., probabilistic) for the input partitions that could not be certified, or repair the analyzed neural network models by eliminating bias. More generally, we believe that the design of the approach and its underlying ideas, such as that of leveraging abstract activation patterns, are potentially useful in other verification settings, e.g., proving functional properties of neural networks [Katz et al. 2017; Wang et al. 2018, etc.].

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their constructive feedback. This work was supported by DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science).

REFERENCES


